9 December 2024

CS205 Final Project Design Manual

Tue Nhi Cao, Meiers Dixon, Chris Garcia, & Molly Yoder

Introduction

The crossword application is an object-oriented system built with Java and JavaFX to dynamically create and manage crosswords for educational and gaming purposes (future development). It features a controller hierarchy for scene navigation and user interactions, supported by utility classes for word validation, hint generation, and crossword construction. Educational Mode allows users to input custom word lists for tailored crosswords, while Game Mode generates puzzles from preloaded databases. Key components like PuzzleGenerator and CallAPI ensure efficient grid generation, optimization, and hint creation, enabling a scalable and maintainable design. While the Game Mode is still under development, it is designed to include advanced interactive features such as real-time input validation, scoring systems, and the ability to save or share completed puzzles.

User Stories

Completed user stories:

1. **As a teacher**, I want to input a list of words so that I can create a customized crossword puzzle for my students. (completed)
2. **As a teacher**, I want the system to automatically generate hints for my crossword so that I don’t have to write them manually. (completed)

3. **As a casual user**, I want to navigate between Educational Mode and Game Mode easily so that I can access the features relevant to my interests.
4. **As a user who likes to test error handling**, I want the application to gracefully handle invalid inputs, such as random symbols or characters, so that it doesn't crash and provides helpful feedback.

Future user stories to be completed:

1. **As a casual user**, I want to play a crossword game where the system automatically generates a random puzzle so that I can enjoy a quick, engaging challenge.
2. **As a casual user**, I want the system to highlight correct and incorrect answers in the crossword grid so that I receive instant feedback on my progress.

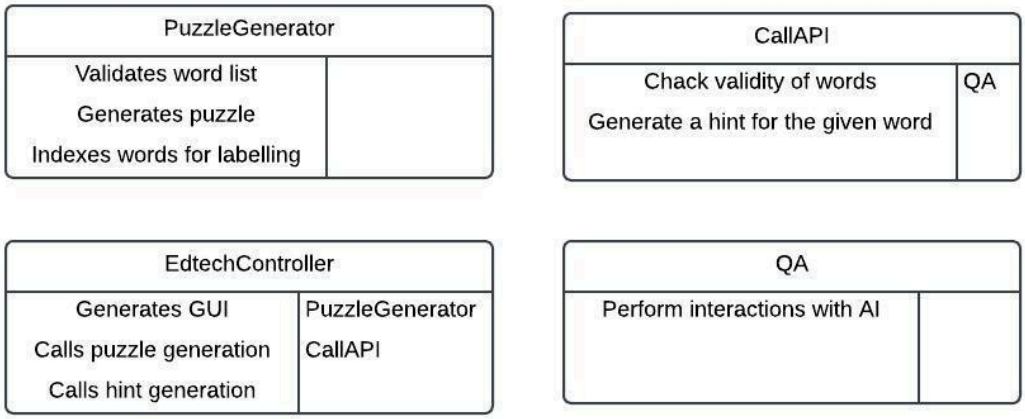
Object-Oriented Design Details

The crossword application employs a scalable object-oriented design (OOD) to manage its functionality and user interface. The application leverages a controller class hierarchy that interacts with JavaFX's Scene Builder, ensuring a seamless user experience.

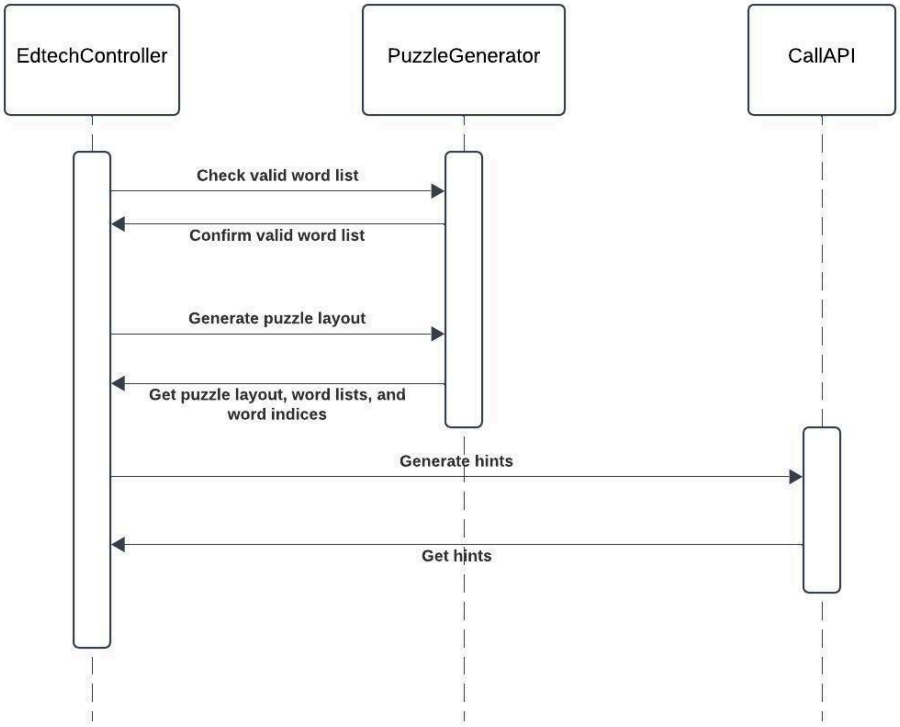
The design is centered around a shared base controller, *BaseController*, which provides common functionality such as managing the main application scene. Derived controllers, such as *EdtechController* and *GameController*, specialize in handling specific sections of the application. The *EdtechController* manages custom crossword generation using user-provided word lists, hint regeneration, input validation, and dynamic updates to the crossword display in Educational Mode. The *GameController* is responsible for generating random crosswords and validating user solutions in Game Mode while the *MainSceneController* oversees navigation between these sections.

Besides some derived classes, we also created some independent classes that have their own responsibilities. PuzzleGenerator validates word lists, dynamically generates crossword grids, and optimizes grid size by trimming unnecessary spaces. CallAPI integrates with an external QA model to generate hints and validate words while the QA class facilitates API communication, processing responses to retrieve hints or validation results.

Below are the CRC cards to further visualize some of the main classes:

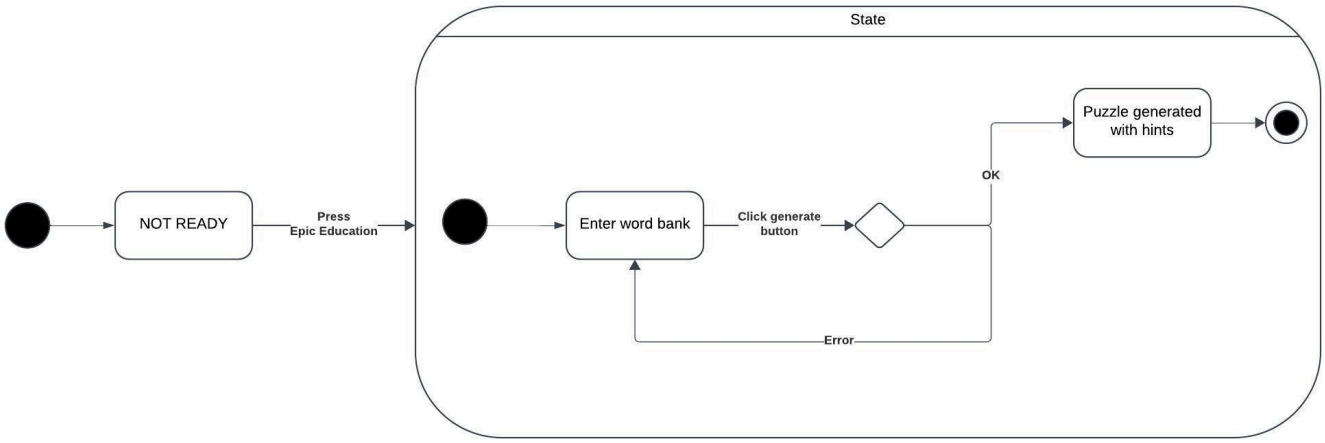


To understand how each class interacts with each other, here is a UML sequence diagram to illustrate the interactions and how information is being delivered in this application:

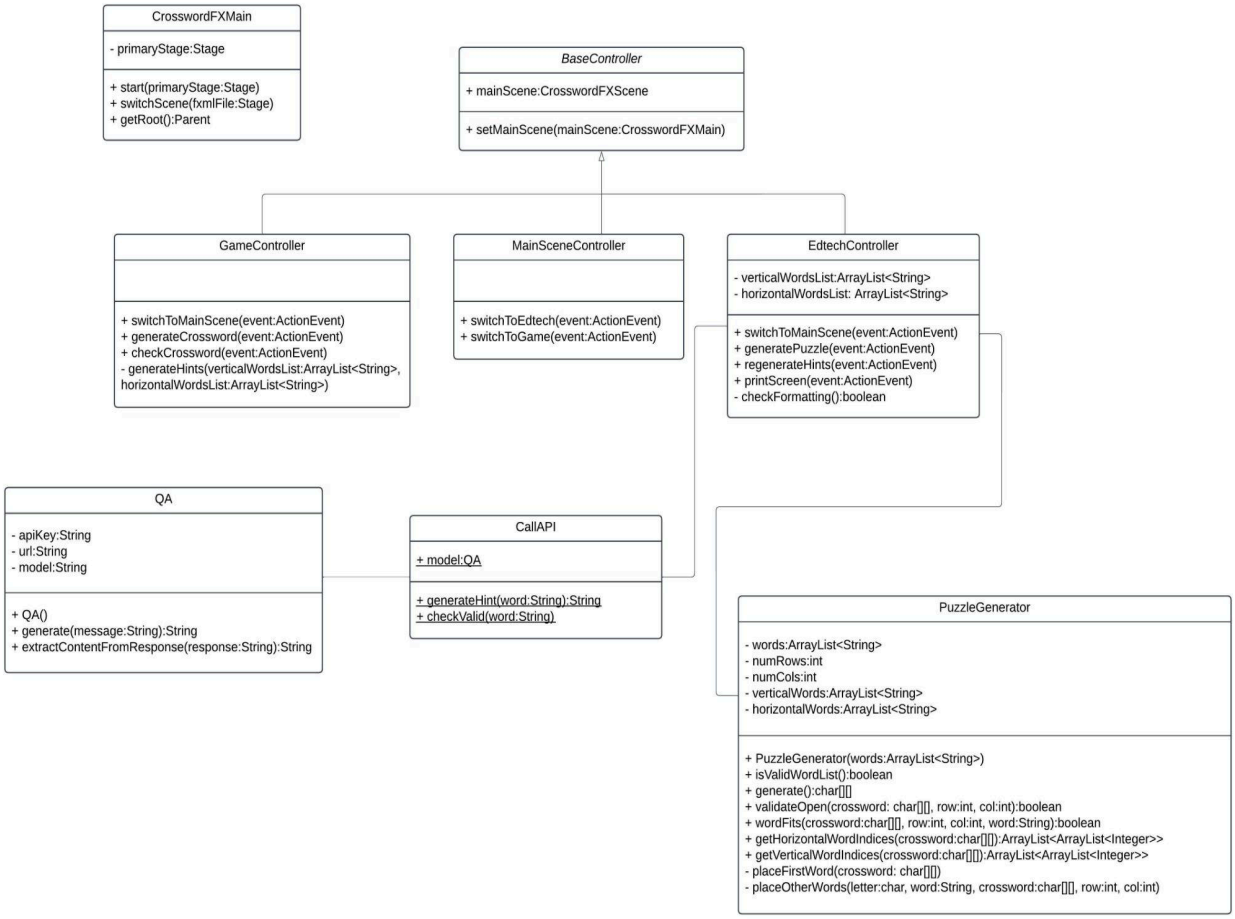


The PuzzleGenerator object is initialized and is used to validate the given word list and build the structure of the puzzle layout. This class primarily contains elements of the word placement algorithm used to determine the puzzle shape. Once this task is complete and all necessary steps are validated, a CallAPI object is initialized which allows us to utilize the power of artificial intelligence to power the generation of hints for each word in the puzzle. Users can regenerate hints as needed for flexibility. Once this is done and the hints are displayed on the GUI, the user has the option to regenerate the hints if they are deemed unsatisfactory. This action is the same as the original hint generation defined in the CallAPI class.

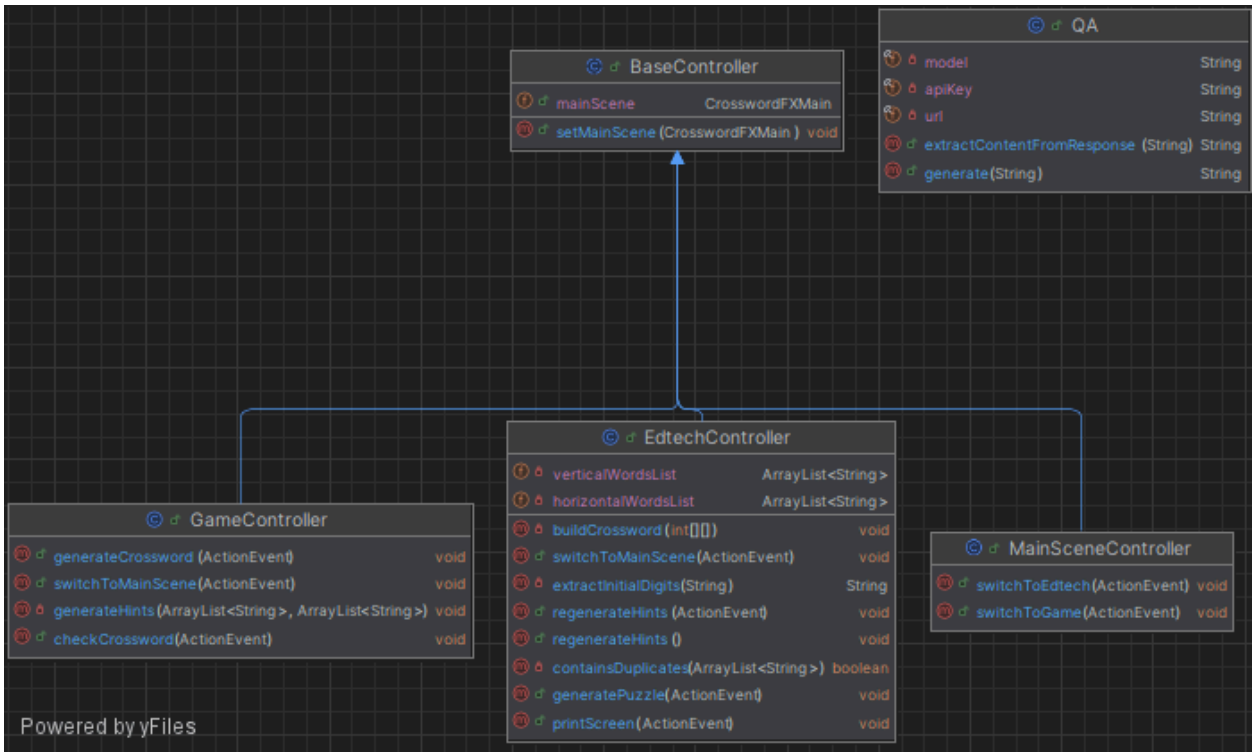
Since our Game Mode is in further development, here is our UML state diagram for the Education Mode below:



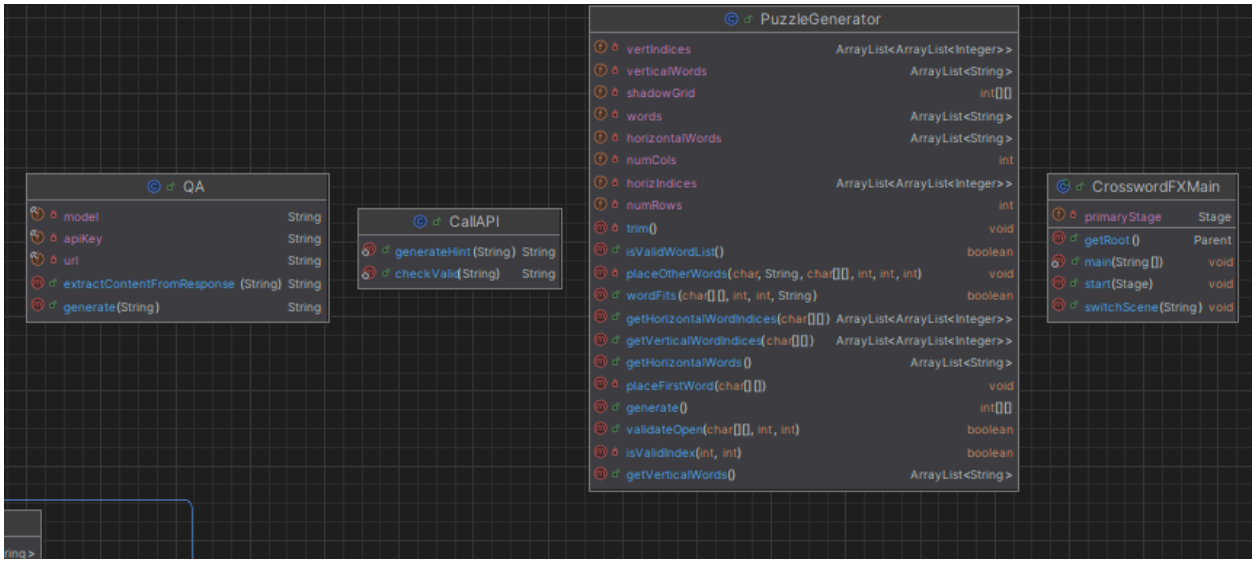
For further information about the variables and which methods we used in each class, we created a high-level UML in Lucid Chart:



Below is a snapshot of the OOP between the controller classes mentioned above:



The picture below shows more detail about how we implemented our PuzzleGenerator and CallAPI classes.



The PuzzleGenerator is responsible for processing input word lists to ensure they are suitable for creating crosswords. It dynamically generates grids by placing words in logical positions, starting with a central placement for the first word and adding subsequent words with

proper alignment and intersections. Core components like a grid tracker and categorized word lists enhance the generation process and support hint creation and gameplay. The QA class integrates an external API to dynamically generate hints and validate words for the application. It handles communication with the API, sending user queries and processing responses to extract relevant results. For further reading, here is the Github repository source that we took a look at before implementing this class:

https://github.com/Ayush-singh141/Code_Alpha_Artificial_Intelligence_Chatbot/blob/main/ChatBot.java